GUI BASED "SECURITY AGENT" FOR LINUX

SYNOPSIS OF MAJOR PROJECT

Submitted for the partial fulfillment of the requirement of Degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY INSTITUTE OF TECHNOLOGY

RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA

BHOPAL- 462033

DEC 2015

Submitted By :-

SHUBHAM SAHU

VIKAS BHADORIYA

VISHAL TIWARI

SHWETA SINGH LODHI

Guided By :-

DR. SHIKHA AGRAWAL

PROF. RAJU BARASKAR

Table of Contents

CHAPTER-1

INTRODUCTION

Overview:

“1st Security Agent” is an excellent password-protected security utility to secure Linux-based computers. It works under any Redhat Linux flavor and offers an administrative support for controlling which users are allowed to access your computer and the level of access each user may have. You can choose to restrict access to lots of Administrative Panel applet functions, including Display, Network, Passwords, Printers, System, Add/Remove Programs, etc. You can also assign separate system profile folders to each user, providing each with their own custom Desktop, Start Menu, Favorites, Documents, etc. Additionally, you can: disable Start Menu items, lock local, network and USB drives, disable the Virtual Console or Terminal, boot keys, BASH mode, config file editing, task manager, and network access, hide desktop icons, and much more. You can apply password protection to Linux users and restrict them to running specific applications only. Security restrictions can be applied universally or just to specific users. "1st Security Agent" also supports Firefox security that enables you to customize many aspects of the Firefox Web browser. It lets you disable individual menu items, prevent others from editing your Favorites, disable individual tabs in the Internet Options dialog, as well as specific settings from each tab. "1st Security Agent" allows you to import and export PC security settings, and offers a flexible and complete password protection. You'll find the program interface very easy to negotiate.

Software Requirement:

- Python 2.7
- PyQt4
- Shell Scripting
- Red Hat Enterprise Linux 7.1
- MySQL (as database)

CHAPTER-2

INTRODUCTION TO LINUX & HIERARCHY ROOT FILE SYSTEM

Linux is a fast and stable open source operating system for personal computers (PCs) and workstations that features professional-level Internet services, extensive development tools, fully functional graphical user interfaces (GUIs), and a massive number of applications ranging from office suites to multimedia applications. Linux was developed in the early 1990s by Linus Torvalds, along with other programmers around the world. As an operating system, Linux performs many of the same functions as Unix, Macintosh, Windows, and Windows NT. However, Linux is distinguished by its power and flexibility, along with being freely available. Most PC operating systems, such as Windows, began their development within the confines of small, restricted PCs, which have only recently become more versatile machines. Such operating systems are constantly being upgraded to keep up with the ever-changing capabilities of PC hardware.

Linux, on the other hand, was developed in a different context. Linux is a PC version of the Unix operating system that has been used for decades on mainframes and minicomputers and is currently the system of choice for network servers and workstations. Linux brings the speed, efficiency, scalability, and flexibility of Unix to your PC, taking advantage of all the capabilities that PCs can now provide. Technically, Linux consists of the operating system program, referred to as the kernel, which is the part originally developed by Linus Torvalds. But it has always been distributed with a massive number of software applications, ranging from network servers and security programs to office applications and development tools. Linux has evolved as part of the open source software movement, in which independent programmers joined together to provide free, high-quality software to any user. Linux has become the premier platform for open source

software, much of it developed by the Free Software Foundation's GNU project. Many of these applications are bundled as part of standard Linux distributions. Currently, thousands of open source applications are available for Linux from sites like SourceForge, Inc.'s sourceforge.net, K Desktop Environment's (KDE's) kde-apps.org, and GNU Network Object Model Environment's (GNOME's) gnomefiles.org. Most of these applications are also incorporated into the distribution repository, using packages that are distribution compliant. Along with Linux's operating system capabilities come powerful networking features, including support for Internet, intranets, and Windows networking. As a norm, Linux distributions include fast, efficient, and stable Internet servers, such as the web, File Transfer Protocol (FTP), and DNS servers, along with proxy, news, and mail servers. In other words, Linux has everything you need to set up, support, and maintain a fully functional network.

Linux

Originally designed specifically for Intel-based PCs, Linux started out at the University of Helsinki as a personal project of a computer science student named Linus Torvalds. At that time, students were making use of a program called Minix,which highlighted different Unix features. Minix was created by Professor Andrew Tanenbaum and widely distributed over the Internet to students around the world. Linus's intention was to create an effective PC version of Unix for Minix users. It was named Linux, and in 1991, Linus released version 0.11. Linux was widely distributed over the Internet, and in the following years, other programmers refined and added to it, incorporating most of the applications and features now found in standard Unix systems. All the major window managers have been ported to Linux. Linux has all the networking tools, such as FTP support, web browsers, and the whole range of network services such as email, the domain name service, and dynamic host configuration, along with FTP, web, and print servers. It also has a full set of program development utilities, such as C++ compilers and debuggers. Given all its features, the Linux operating system remains small, stable, and fast. In its simplest format, Linux can run effectively on only 2MB of memory.

Although Linux has developed in the free and open environment of the Internet, it adheres to official Unix standards. Because of the proliferation of Unix versions in the previous decades, the Institute of Electrical and Electronics Engineers (IEEE) developed an independent Unix standard for the American National Standards Institute (ANSI). This new ANSI-standard Unix is called the Portable Operating System Interface for Computer Environments (POSIX). The standard defines how a Unix-like system needs to operate, specifying details such as system calls and interfaces. POSIX defines a universal standard to which all Unix versions must adhere. Most popular versions of Unix are now POSIX-compliant. Linux was developed from the beginning according to the POSIX standard. Linux also adheres to the Linux file system hierarchy standard (FHS), which specifies the location of files and directories in the Linux file structure.Seepathname.com/fhsfor more details.

Linux Overview

Like Unix, Linux can be generally divided into three major components: the kernel, the environment, and the file structure. The kernel is the core program that runs programs and manages hardware devices, such as disks and printers. The environment provides andinterface for the user. It receives commands from the user and sends those commands to the kernel for execution. The file structure organizes the way files are stored on a storage device, such as a disk. Files are organized into directories. Each directory may contain any number of subdirectories, each holding files. Together, the kernel, the environment, and the file structure form the basic operating system structure. With these three, you can run programs, manage files, and interact with the system.

An environment provides an interface between the kernel and the user. It can be described as an interpreter. Such an interface interprets commands entered by the user and sends them to the kernel. Linux provides several kinds of environments: desktops, window managers, and command line shells. Each user on a Linux system has his or her own user interface. Users can tailor their environments to their own special needs, whether they be shells, window managers,

or desktops. In this sense, for the user, the operating system functions more as an operating environment, which the user can control.In Linux, files are organized into directories, much as they are in Windows. The entire Linux file system is one large interconnected set of directories, each containing files. Some directories are standard directories reserved for system use. You can create your own directories for your own files, as well as easily move files from one directory to another. You can even move entire directories and share directories and files with other users on your system. With Linux, you can also set permissions on directories and files, allowing others to access them or restricting access to yourself alone. The directories of each user are, in fact, ultimately connected to the directories of other users. Directories are organized into a hierarchical tree structure, beginning with an initial root directory. All other directories are ultimately derived from this first root directory.

LINUX DESKTOPS

With KDE & GNOME, Linux now has a completely integrated GUI. You can perform all your Linux operations entirely from either interface. KDE & GNOME are fully operational desktops supporting drag-and-drop operations, enabling you to drag icons to your desktop and to set up your own menus on an Applications panel. Both rely on an underlying X Window System, which means as long as they are both installed on your system, applications from one can run on the other desktop. The GNOME and KDE sites are particularly helpful for documentation, news, and software you can download for those desktops. Both desktops can run any X Window System program, as well as any cursor-based program such as Emacs and Vi, which were designed to work in a shell environment. At the same time, a great many applications are written just for those desktops and included with your distributions. KDE & GNOME have complete sets of Internet tools, along with editors and graphics, multimedia, and system applications. Check their websites at gnome.org and kde.org for latest developments. As new versions are released, they include new software.

Open Source Software

Linux was developed as a cooperative open source effort over the Internet, so no company or institution controls Linux. Software developed for Linux reflects this background. Development often takes place when Linux users decide to work on a project together. The software is posted at an Internet site, and any Linux user can then access the site and download the software. Linux software development has always operated in an Internet environment and is global in scope, enlisting programmers from around the world. The only thing you need to start a Linux-based software project is a website.Most Linux software is developed as open source software. This means that the source code for an application is freely distributed along with the application. Programmers over the Internet can make their own contributions to a software package's development, modifying and correcting the source code. Linux is an open source operating system as well. Its source code is included in all its distributions and is freely available on the Internet. Many major software development efforts are also open source projects, as are the KDE & GNOME desktops, along with most of their applications. The Netscape Communicator web browser package has also become open source, with its source code freely available. The Open Office office suite supported by Sun is an open source project based on the StarOffice office suite (StarOffice is essentially Sun's commercial version of OpenOffice). Many of the open sourceapplications that run on Linux have located their websites at SourceForge (sourceforge.net), which is a hosting site designed specifically to support open source projects. You can find more information about the open source movement at opensource.org.Open source software is protected by public licenses. These prevent commercial companies from taking control of open source software by adding a few modifications of their own, copy righting those changes, and selling the software as their own product.

The most popular public license is the GNU GPL provided by the Free Software Foundation. This is the license that Linux is distributed under. The GNU GPL retains the copyright, freely licensing the software with the requirement that the software and any modifications made to it always be freely available. Other public licenses have also been created to support the demands

of different kinds of open source projects. The GNU lesser general public license (LGPL) lets commercial applications use GNU licensed software libraries. The qt public license (QPL) lets open source developers use the Qt libraries essential to the KDE desktop. You can find a complete listing at opensource.org.

Linux is currently copyrighted under a GNU public license provided by the Free Software Foundation, and it is often referred to as GNU software (seegnu.org). GNU software is distributed free, provided it is freely distributed to others. GNU software has proved both reliable and effective. Many of the popular Linux utilities, such as C compilers, shells, and editors, are GNU software applications. Installed with your Linux distribution are the GNU C++ and Lisp compilers, Vi and Emacs editors, BASH and TCSH shells, as well as TeX and Ghostscript document formatters. In addition, there are many open source software projects that are licensed under the GNU GPL.Under the terms of the GNU GPL, the original author retains the copyright, although anyone can modify the software and redistribute it, provided the source code is included, made public, and provided free. Also, no restriction exists on selling the software or giving it away free. One distributor could charge for the software, while another one could provide it free of charge. Major software companies are also providing Linux versions of their most popular applications. Oracle provides a Linux version of its Oracle database. (At present, no plans seem in the works for Microsoft applications.)

LINUX DISTRIBUTIONS

Red Hat Linux	redhat.com
Fedora Linux	fedoraproject.org
Centos Linux	centos.org
OpenSUSE Linux	opensuse.com
Debian Linux	debian.org
Ubuntu Linux	ubuntu.com
Mepis Linux	mepis.org
Gentoo Linux	gentoo.org

Turbo Linux	turbolinux.c

LINUX FILESYSTEM

Root Directory: /

The subdirectories held in the root directory, /, are listed in Table 29-1, along with other useful subdirectories. Directories that you may commonly access as an administrator are the /etc directory, which holds configuration files; the /dev directory, which holds dynamically generated device files; and the /vardirectory, which holds server data files for DNS, web, mail, and FTP servers, along with system logs and scheduled tasks. For managing different versions of the kernel, you may need to access the /boot and /lib/modules directories as well as /usr/src/linux. The /boot directory holds the kernel image files for any new kernels you install, and the /lib/modules directory holds modules for your different kernels

The /usr Directory

The /usr directory contains a multitude of important subdirectories used to support users,providing applications, libraries, and documentation. The /usr/bin directory holds numerous user-accessible applications and utilities; /usr/sbin hold user-accessible administrative utilities. The /usr/share directory holds architecture-independent data that includes an extensive number of subdirectories, including those for documentation, such as man, info, and doc files.

The /media Directory

The /media directory is used for mountpoints for removable media like CD-ROM, DVD, floppy, or Zip drives, as well as for other media-based file systems such as USB card readers, cameras, and MP3 players. These are file systems you may be changing frequently,unlike partitions on fixed disks. Most Linux systems use the Hardware Abstraction Layer (HAL) to dynamically manage the creation, mounting, and device assignment of these devices. As instructed by HAL, this tool will create floppy, CD-ROM, storage card, camera, and MP3 player subdirectories in

/media as needed. The default subdirectory for mounting is /media/disk. Additional drives have an number attached to their name.

The /mnt Directory

The /mnt directory is usually used for mount points for other mounted file systems such as Windows partitions. You can create directories for any partitions you want to mount, such as /mnt/windows for a Windows partition.

The /home Directory

The /home directory holds user home directories. When a user account is set up, a home directory is set up here for that account, usually with the same name as the user.As the system administrator, you can access any user's home directory, so you have control over that user's files.

The /var Directory

The /vardirectory holds subdirectories for tasks whose files change frequently, such as lock files, log files, web server files, or printer spool files. For example, the /vardirectory holds server data directories, such as /var/www for the Apache web server website files or /var/ftp for your FTP site files, as well as /var/named for the DNS server. The /tmp directory is simply a directory to hold any temporary files programs may need to perform a particular task.The /var directories are designed to hold data that changes with the normal operation of the Linux system. For example, spool files for documents that you are printing are kept here. A spool file is created as a temporary printing file and is removed after printing. Other files, such as system log files, are changed constantly.

The /proc System

The /proc file system is a special file system that is generated in system memory. It does not exist on any disk. /proc contains files that provide important information about the state of your system. For example, /proc/cpuinfo holds information about your computer's CPU processor, /proc/devices lists those devices currently configured to run with your kernel, /proc/filesystems lists the file systems, and /proc files are really interfaces to the kernel, obtaining information from the kernel about your system. Table 29-5 lists the /proc subdirectories and files.like any file system, /prochas to be mounted. The /etc/fstab file will have a special entry for /proc with a file system type of proc and no device specified.

The sysfs File System: /sys

The sysfs file system is a virtual file system that provides the a hierarchical map of your kernel-supported devices such as PCI devices, buses, and block devices, as well as supporting kernel modules. The classes subdirectory will list all your supported devices by device file a particular device is associated with. This is very helpful for managing removable devices as well as dynamically configuring and managing devices as HAL and udev do. The sysfs file system is used by udev to dynamically generate needed device files in the /dev directory, as well as by HAL to manage removable device files and support as needed (HAL technically provides information only about devices, though it can use tools to dynamically change configurations as needed). The /sys filesystem type is sysfs. The /sys subdirectories organize your devices into different categories. The file system is used by systool to display a listing of your installed devices.

CHAPTER-3

INTRODUCTION TO PYTHON, PyQt, SHELL SCRIPT

Python:

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

Python interpreters are available for installation on many operating systems, allowing Python code execution on a wide variety of systems. Using third-party tools, such as Py2exe or Pyinstaller, Python code can be packaged into stand-alone executable programs for some of the most popular operating systems, allowing the distribution of Python-based software for use on those environments without requiring the installation of a Python interpreter.

CPython, the reference implementation of Python, is free and open-source software and has a community-based development model, as do nearly all of its alternative implementations. CPython is managed by the non-profit Python Software Foundation.

Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. There are interfaces to many system calls and libraries, as well as to various windowing systems. New built-in modules are easily written in C or C++ (or other languages, depending on the chosen implementation). Python is

also usable as an extension language for applications written in other languages that need easy-to-use scripting or automation interfaces.

History

Python was conceived in the late 1980s, and its implementation was started in December 1989 by Guido van Rossum at CWI in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing with the Amoeba operating system. Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, benevolent dictator for life (BDFL).

Features and philosophy

Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and there are a number of language features which support functional programming and aspect-oriented programming (including by metaprogramming] and by magic methods). Many other paradigms are supported using extensions, including design by contract and logic programming.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. An important feature of Python is dynamic name resolution (late binding), which binds method and variable names during program execution.

The design of Python offers some support for functional programming in the Lisp tradition. The language has map(), reduce() and filter() functions; comprehensions for lists,dictionaries, and sets; and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The core philosophy of the language is summarized by the document "PEP 20 (The Zen of Python)", which includes aphorisms such as:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Indentation

Python uses whitespace indentation, rather than curly braces or keywords, to delimit blocks; this feature is also termed the off-side rule. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block.

Libraries

Python has a large standard library, commonly cited as one of Python's greatest strengths, providing tools suited to many tasks. This is deliberate and has been described as a "batteries included" Python philosophy. For Internet-facing applications, a large number of standard formats and protocols (such as MIME and HTTP) are supported. Modules for creating graphical user interfaces, connecting to relational databases, pseudorandom number generators, arithmetic with arbitrary precision decimals, manipulating regular expressions, and doing unit testing are also included.

Some parts of the standard library are covered by specifications (for example, the WSGI implementation wsgiref follows PEP 333[1]), but the majority of the modules are not. They are specified by their code, internal documentation, and test suite (if supplied). However, because most of the standard library is cross-platform Python code, there are only a few modules that must be altered or completely rewritten by alternative implementations.

Popular Implementations

CPython

This is the original and most-maintained implementation of Python, written in C. New language features generally appear here first.

Jython

Python implemented in Java. This implementation can be used as a scripting language for Java applications, or can be used to create applications using the Java class libraries. It is also often used to create tests for Java libraries. More information can be found at the Jython website.

Python for .NET

This implementation actually uses the CPython implementation, but is a managed .NET application and makes .NET libraries available. It was created by Brian Lloyd. For more information, see the Python for .NET home page.

IronPython

An alternate Python for .NET. Unlike Python.NET, this is a complete Python implementation that generates IL, and compiles Python code directly to .NET assemblies. It was created by Jim Hugunin, the original creator of Jython. For more information, see the IronPython website.

PyPy

An implementation of Python written completely in Python. It supports several advanced features not found in other implementations like stackless support and a Just in Time compiler. One of the goals of the project is to encourage experimentation with the

language itself by making it easier to modify the interpreter (since it is written in Python). Additional information is available on the PyPy project's home page.

Each of these implementations varies in some way from the language as documented in this manual, or introduces specific information beyond what's covered in the standard Python documentation. Please refer to the implementation-specific documentation to determine what else you need to know about the specific implementation you're using.

Reasons to use Python

Readability

Python very closely resembles the English language, using words like 'not' and 'in' to make it to where you can very often read a program, or script, aloud to someone else and not feel like you're speaking some arcane language. This is also helped by Python's very strict punctuation rules which means you don't have curly braces ({ }) all over your code. Also, Python has a set of rules, known as PEP 8, that tell every Python developer how to format their code. This means you always know where to put new lines and, more importantly, that pretty much every other Python script you pick up, whether it was written by a novice or a seasoned professional, will look very similar and be just as easy to read. The fact that my Python code, with five or so years of experience, looks very similar to the code that Guido van Rossum (the creator of Python) writes is such an ego boost.

Libraries

Python has been around for over 20 years, so a lot of code written in Python has built up over the decades and, being an open source language, a lot of this has been released for others to use. Almost all of it is collected on "https://pypi.python.org", pronounced "pie-pee-eye" or, more commonly called "the Cheese Shop". You can install this software on your system to be used by your own projects. For example, if you want to

use Python to build scripts with command line arguments, you'd install the "click" library and then import it into your scripts and use it. There are libraries for pretty much any use case you can come up with, from image manipulation, to scientific calculations, to server automation.

Community

Python has user groups everywhere, usually called PUGs, and does major conferences on every continent other than Antarctica. PyCon NA, the largest Python conference in North America, sold out its 2,500 tickets this year. And, reflecting Python's commitment to diversity, it had over 30% women speakers. PyCon NA 2013 also started a trend of offering "Young Coder" workshops, where attendees taught Python to kids between 9 and 16 years of age for a day, getting them familiar with the language and, ultimately, helping them hack and mod some games on the Raspberry Pis they were given. Being part of a such a positive community does a lot to keep you motivated.

I'm very excited to be able to share my favorite language with the Treehouse community and hopefully the pieces of Python that I love the most will help you decide to check it out and learn it with me.

SHELL SCRIPT:

A shell script is a computer program designed to be run by the Unix shell, a command line interpreter. The various dialects of shell scripts are considered to be scripting languages.

The typical Unix/Linux/Posix-compliant installation includes the Korn Shell (ksh) in several possible versions such as ksh88, Korn Shell '93 and others. The oldest shell still in common use is the Bourne shell (sh); Unix systems invariably include also the C Shell (csh), Bourne Again Shell (bash), a remote shell (rsh), a secure shell for SSL telnet connections (ssh), and a shell which is a main component of the Tcl/Tk installation usually called tclsh; wish is a GUI-based Tcl/Tk shell. Other shells available on a machine or available for download and/or purchase include ash, msh, ysh, zsh (a particularly common enhanced Korn Shell), the Tenex C Shell (tcsh), a Perl-like shell (psh) and others. Related programmes such as shells based on Python, Ruby, C, Java, Perl, Pascal, Rexx &c in various forms. Another somewhat common shell is osh, whose manual page states it "is an enhanced, backward-compatible port of the standard command interpreter from Sixth Edition UNIX."

Interoperability software such as Cygwin, the MKS Toolkit, Interix (which is available in the Microsoft Windows Services for Unix), Hamilton C shell, UWIN (AT&T Unix for Windows) and others allow Unix shell programmes to be run on machines running Windows NT and its successors, with some loss of functionality on the MS-DOS-Windows 95 branch, as well as earlier MKS Toolkit versions for OS/2. At least three DCL implementations for Windows type operating systems -- in addition to XLNT, a multiple-use scripting language package which is used with the command shell, Windows Script Host and CGI programming -- are available for these systems as well. Mac OS X and subsequent are Unix-like as well.

In addition to the aforementioned tools, some Posix and OS/2 functionality can be used with the corresponding environmental subsystems of the Windows NT operating system series up to Windows 2000 as well. A third, 16-bit subsystem often called the MS-DOS subsystem uses the Command.com provided with these operating systems to run the aforementioned MS-DOS batch files.

The console alternatives 4NT, 4DOS, 4OS2, and the GUI Take Command which add functionality to the Windows NT-style Cmd.exe, MS-DOS/Windows 95 batch files (run by

Command.com), OS/2's Cmd.exe, and 4NT respectively are similar to the shells that they enhance and are more integrated with the Windows Script Host, which comes with three pre-installed engines, VBScript, JScript, and VBA and to which numerous third-party engines can be added, with Rexx, Perl, Python, Ruby, and Tcl having pre-defined functions in 4NT and related programmes. PC DOS is quite similar to MS-DOS, whilst DR DOS is more different. Earlier versions of Windows NT are able to run contemporary versions of 4OS2 by the OS/2 subsystem.

Scripting languages are, by definition, able to be extended; for example, a MS-DOS/Windows 95/98 and Windows NT type systems allows for shell/batch programmes to call tools like KixTart, QBasic, various Basic, Rexx, Perl, and Python implementations, the Windows Script Host and its installed engines. On Unix and other Posix-compliant systems, awk and sed are used to extend the string and numeric processing ability of shell scripts. Tcl, Perl, Rexx, and Python have graphics toolkits and can be used to code functions and procedures for shell scripts which pose a speed bottleneck (C, Fortran, assembly language &c are much faster still) and to add functionality not available in the shell language such as sockets and other connectivity functions, heavy-duty text processing, working with numbers if the calling script does not have those abilities, self-writing and self-modifying code, techniques like recursion, direct memory access, various types of sorting and more, which are difficult or impossible in the main script, and so on. Visual Basic for Applications and VBScript can be used to control and communicate with such things as spreadsheets, databases, scriptable programmes of all types, telecommunications software, development tools, graphics tools and other software which can be accessed through the Component Object Model.

Typical operations performed by shell scripts include file manipulation, program execution, and printing text. A script which sets up the environment, runs the programme, and does any necessary cleanup, logging, &c is called a wrapper.

Capabilities

- ### Shortcuts

  A shell script can provide a convenient variation of a system command where special environment settings, command options, or post-processing apply automatically, but in a way that allows the new script to still act as a fully normal Unix command.

  One example would be to create a version of ls, the command to list files, giving it a shorter command name of l, which would be normally saved in a user's bin directory as/home/*username*/bin/l, and a default set of command options pre-supplied.

  ```
  #!/bin/sh
  LC_COLLATE=C ls -FCas "$@"
  ```

  Here, the first line (shebang) indicates which interpreter should execute the rest of the script, and the second line makes a listing with options for file format indicators, columns, all files (none omitted), and a size in blocks. The LC_COLLATE=C sets the default collation order to not fold upper and lower case together, not intermix dotfiles with normal filenames as a side effect of ignoring punctuation in the names (dotfiles are usually only shown if an option like -a is used), and the "$@" causes any parameters given to l to pass through as parameters to ls, so that all of the normal options and other syntax known to ls can still be used.
  The user could then simply use l for the most commonly used short listing.
  Another example of a shell script that could be used as a shortcut would be to print a list of all the files and directories within a given directory.

  ```
  #!/bin/sh
  clear
  ls -al
  ```

In this case, the shell script would start with its normal starting line of #!/bin/sh. Following this, the script executes the command clear which clears the terminal of all text before going to the next line. The following line provides the main function of the script. The ls -al command list the files and directories that are in the directory from which the script is being run. The ls command attributes could be changed to reflect the needs of the user.

Note: If an implementation does not have the clear command, try using the 'clr' command instead.

- **Batch jobs**

  Shell scripts allow several commands that would be entered manually at a command-line interface to be executed automatically, and without having to wait for a user to trigger each stage of the sequence. For example, in a directory with three C source code files, rather than manually running the four commands required to build the final program from them, one could instead create a C shell script, here named build and kept in the directory with them, which would compile them automatically:

```
#!/bin/csh
echo compiling...
cc -c foo.c
cc -c bar.c
cc -c qux.c
cc -o myprog foo.o bar.o qux.o
echo done.
```

  The script would allow a user to save the file being edited, pause the editor, and then just run ./build to create the updated program, test it, and then return to the editor. Since the 1980s or so, however, scripts of this type have been replaced with utilities like make which are specialized for building programs.

- **Generalization**

  Simple batch jobs are not unusual for isolated tasks, but using shell loops, tests, and variables provides much more flexibility to users. A Bash (Unix shell) script to convert JPEG images to PNG images, where the image names are provided on the command line—possibly via wildcards—instead of each being listed within the script, can be created with this file, typically saved in a file like /home/username/bin/jpg2png

```
#!/bin/bash
for jpg; do                          # use $jpg in place of each filename given, in turn
   png="${jpg%.jpg}.png"             # construct the PNG version of the filename by replacing .jpg with .png
   echo converting "$jpg" ...        # output status info to the user running the script
   if convert "$jpg" jpg.to.png ; then   # use the convert program (common in Linux) to create the PNG in a temp file
      mv jpg.to.png "$png"           # if it worked, rename the temporary PNG image to the correct name
   else                              # ...otherwise complain and exit from the script
      echo 'jpg2png: error: failed output saved in "jpg.to.png".' >&2
      exit 1
   fi                                # the end of the "if" test construct
done                                 # the end of the "for" loop
echo all conversions successful      # tell the user the good news
exit 0
```

The jpg2png command can then be run on an entire directory full of JPEG images with just /home/*username*/bin/jpg2png *.jpg

- **Verisimilitude**

A key feature of shell scripts is that the invocation of their interpreters is handled as a core operating system feature. So rather than a user's shell only being able to execute scripts in that shell's language, or a script only having its interpreter directive handled correctly if it was run from a shell (both of which were limitations in the early Bourne shell's handling of scripts), shell scripts are set up and executed by the OS itself. A modern shell script is not just on the same footing as system commands, but rather many system commands are actually shell scripts (or more generally, scripts, since some of them are not interpreted by a shell, but instead by Perl, Python, or some other language). This extends to returning exit codes like other system utilities to indicate success or failure, and allows them to be called as components of larger programs regardless of how those larger tools are implemented.

Like standard system commands, shell scripts classically omit any kind of filename extension unless intended to be read into a running shell through a special mechanism for this purpose (such as sh's "**.** ", or csh's source).

- **Programming**

Many modern shells also supply various features usually found only in more sophisticated general-purpose programming languages, such as control-flow constructs, variables,comments, arrays, subroutine and so on. With these sorts of features available, it is possible to write reasonably sophisticated applications as shell scripts. However, they are still limited by the fact that most shell languages have little or no support for data typing systems, classes, threading, complex math, and other common full language features, and are also generally much slower than compiled code or interpreted languages written with speed as a performance goal.

PyQt:

PyQt is a Python binding of the cross-platform GUI toolkit Qt. It is one of Python's options for GUI programming. Popular alternatives are PySide (the Qt binding with official support and a more liberal licence), PyGTK, wxPython, and Tkinter (which is bundled with Python). Like Qt, PyQt is free software. PyQt is implemented as a Python plug-in.

PyQt is developed by the British firm Riverbank Computing. It is available under similar terms to Qt versions older than 4.5; this means a variety of licenses including GNU General Public License (GPL) and commercial license, but not the GNU Lesser General Public License (LGPL). PyQt supports Microsoft Windows as well as various flavours of Unix, including Linux and OS X.

PyQt implements around 440 classes and over 6,000 functions and methods including:

- a substantial set of GUI widgets
- classes for accessing SQL databases (ODBC, MySQL, PostgreSQL, Oracle, SQLite)
- QScintilla, Scintilla-based rich text editor widget
- data aware widgets that are automatically populated from a database
- an XML parser
- SVG support
- classes for embedding ActiveX controls on Windows (only in commercial version)

To automatically generate these bindings, Phil Thompson developed the tool SIP, which is also used in other projects.

In August 2009, Nokia, the then owners of the Qt toolkit, released PySide, providing similar functionality, but under the LGPL, after failing to reach an agreement with Riverbank Computing to change its licensing terms to include LGPL as an alternative license.

PyQt main components:

PyQt4 contains the following Python modules.

- ***QtCore***: The QtCore module contains the core non-GUI classes, including the event loop and Qt's signal and slot mechanism. It also includes platform independent abstractions for Unicode, threads, mapped files, shared memory, regular expressions, and user and application settings.
- ***QtGui***: The QtGui module contains the majority of the GUI classes. These include a number of table, tree and list classes based on the model–view–controller design pattern. Also provided is a sophisticated 2D canvas widget capable of storing thousands of items including ordinary widgets.
- ***QtNetwork***: The QtNetwork module contains classes for writing UDP and TCP clients and servers. It includes classes that implement FTP and HTTP clients and support DNS lookups. Network events are integrated with the event loop making it very easy to develop networked applications.
- ***QtOpenGL***: The QtOpenGL module contains classes that enable the use of OpenGL in rendering 3D graphics in PyQt applications.
- ***QtSql***: The QtSql module contains classes that integrate with open-source and proprietary SQL databases. It includes editable data models for database tables that can be used with GUI classes. It also includes an implementation of SQLite.
- ***QtSvg***: The QtSvg module contains classes for displaying the contents of SVG files. It supports the static features of SVG 1.2 Tiny.
- ***QtXml***: The QtXml module implements SAX and DOM interfaces to Qt's XML parser.
- ***QtMultimedia***: The QtMultimedia module implements low-level multimedia functionality. Application developers would normally use the phonon module.
- ***QtDesigner***: The QtDesigner module contains classes that allow Qt Designer to be extended using PyQt.

- ***Qt***: The Qt module consolidates the classes contained in all of the modules described above into a single module. This has the advantage that you don't have to worry about which underlying module contains a particular class. It has the disadvantage that it loads the whole of the Qt framework, thereby increasing the memory footprint of an application. Whether you use this consolidated module, or the individual component modules is down to personal taste.
- ***uic***: The uic module implements support for handling the XML files created by Qt Designer that describe the whole or part of a graphical user interface. It includes classes that load an XML file and render it directly, and classes that generate Python code from an XML file for later execution.

The QtCore module contains the core non GUI functionality. This module is used for working with time, files and directories, various data types, streams, URLs, mime types, threads or processes. The QtGui module contains the graphical components and related classes. These include for example buttons, windows, status bars, toolbars, sliders, bitmaps, colours, and fonts. The QtNetwork module contains the classes for network programming. These classes facilitate the coding of TCP/IP and UDP clients and servers by making the network programming easier and more portable. The QtXml contains classes for working with XML files. This module provides implementation for both SAX and DOM APIs. The QtSvg module provides classes for displaying the contents of SVG files. Scalable Vector Graphics (SVG) is a language for describing two-dimensional graphics and graphical applications in XML. The QtOpenGL module is used for rendering 3D and 2D graphics using the OpenGL library. The module enables seamless integration of the Qt GUI library and the OpenGL library. The QtSql module provides classes for working with databases.

PyQt exposes much of the functionality of Qt to Python, including:

- A comprehensive set of widgets
- Flexible layout managers

- Standard GUI features for applications (menus, toolbars, dock windows)
- Easy communication between application components (signals and slots)
- A unified painting system with transparency, antialiasing, OpenGL integration and SVG support
- Internationalization (i18n) support and integration with the Qt Linguist translation tool
- Rich text processing, display and printing facilities, including support for PDF export (Qt 4.1 and later)
- Database support (SQL) and model/view features
- Threading classes, providing abstractions for threads, mutexes and semaphores
- Integrated resource handling for applications
- Widget styles, including support for widget style sheets (from Qt 4.2 onwards)
- Input/output and networking
    - Qt 4.3 and later supports Secure Sockets Layer communications
- A powerful, feature-rich graphics canvas (Qt 4.2 and later)
- Inter-process communication for Unix via D-Bus message buses (Qt 4.2 and later)
- Text completion and undo frameworks (Qt 4.2 and later)
- XML handling APIs, including SAX, DOM and (from Qt 4.3 onwards) a stream-oriented API
- Support for XML query technologies, such as XQuery, XPath and XSLT.
- An ECMAScript interpreter with the ability to access Qt's object model
- Support for Qt Designer, including facilities to add custom Python-based widgets to the standard set of Qt widgets available to user interface designers.
- WebKit browser engine integration.
- Support for audio and video playback.

CHAPTER-4

PROBLEM DESCRIPTION

The protocols and methods used to implement and manage security applications and other Linux chores are way too labyrinthe or vastly distributed among several independent software units. So overall, there is no such single software module to accommodate management, or implementation of security. For example, if one needs to improve security of one's Linux computer, one has to implement each and every detail like Firewall, or Intrusion Detection or Access control or any such restrictions manually which is tedious and complex task for one such rookie.

Linux is already a infamous and complex platform when it comes to user friendliness or GUI support, in comparison to Windows and Mac. This is just because of the mere fact that each and every function and feature provided in linux is too flexible and deep for a normal or non-technical user. However, when learnt with sophistication and in great detail, Linux provides the best features, support and functionality in class. Even though Linux is infamous among beginners, craze is increasing.

"1st Security Agent" is developed keeping all such problems in mind, and to overcome the problem of feature distribution. "1st Security Agent" provides a user-friendly GUI platform, where even a beginner could exhaust the functionality of Linux in depth. This project will help to increase popularity of Linux among computer science students and beginners, and help one to learn and understand Linux with better approach.

CHAPTER-5

PROPOSED WORK

“1st Security Agent” is integrated Linux functionality and Security Agent, which means almost everything related to Server, Security and Advance management will be provided at single place.

"1st Security Agent" provides easy way to configure and modify Linux Security like configuring firewall and IDS (Intrusion Detection System) or ACL via user-friendly GUI. Thus "1st Security Agent", reduce the overhead of learning Linux in depth prior to using it.

In general we propose following features under "1st Security Agent":

- ACL configuration
- Firewall Configuration
- DNS security management
- DHCP management
- FTP security management and configuration
- SSH security management
- Packet sniffing
- Packet analysis
- Sudoers management
- Package Control
- User Restrictions
- Backup & Restore
- Mounting of various hardware devices
- Runlevel management

All these features under one roof.

CHAPTER-6

EXPECTED OUTCOME

We will do our best to add all the security features within our knowledge and to make it a complete integrated tool to allow even a novice user in LINUX to easily manage and make the system safe from all the day to day vulnerabilities present today. This tool will not only safeguard the network but also free the user from the burden of remembering the commands. "1st Security Agent" has been designed while keeping the user's security issue as its prime objective. The tool also provides a graphical interface for the user which automatically runs the specific set of commands in an automated fashion . The user only need to have a look at the security issues he is facing and he may face in the near future.

Predicted Features:

- Layman Firewall configuration
- Pre-packaged “Intrusion Detection System”
- In-built packet sniffer and analyser
- Package Control (Add/remove preinstalled software)
- Access Control List (ACL of private/public user directory and files)
- Scheduling
- Easy and rookie backup and restore management
- Complete Server support management and configuration
- History management
- Mount point management (/etc/fstab)
- Runlevel Management (/etc/inittab)
- User based Virtual Console restriction (/etc/securetty)
- Pluggable Authentication Module (PAM) configuration
- Sudoers Control
- Desktop Selection (GNOME/KDE)

We are looking forward to merge in this tool other features like PACKET ANALYSER and INTRUSION DETECTION AND RECOVERY SYSTEM. These tools will provide the functionality of INCOMING AND OUTGOING traffic analysis based on the protocols they use port numbers. And they will also allow the user to know about the attacks on his system in case one occurs.

For future work we hope to add following functionalities:

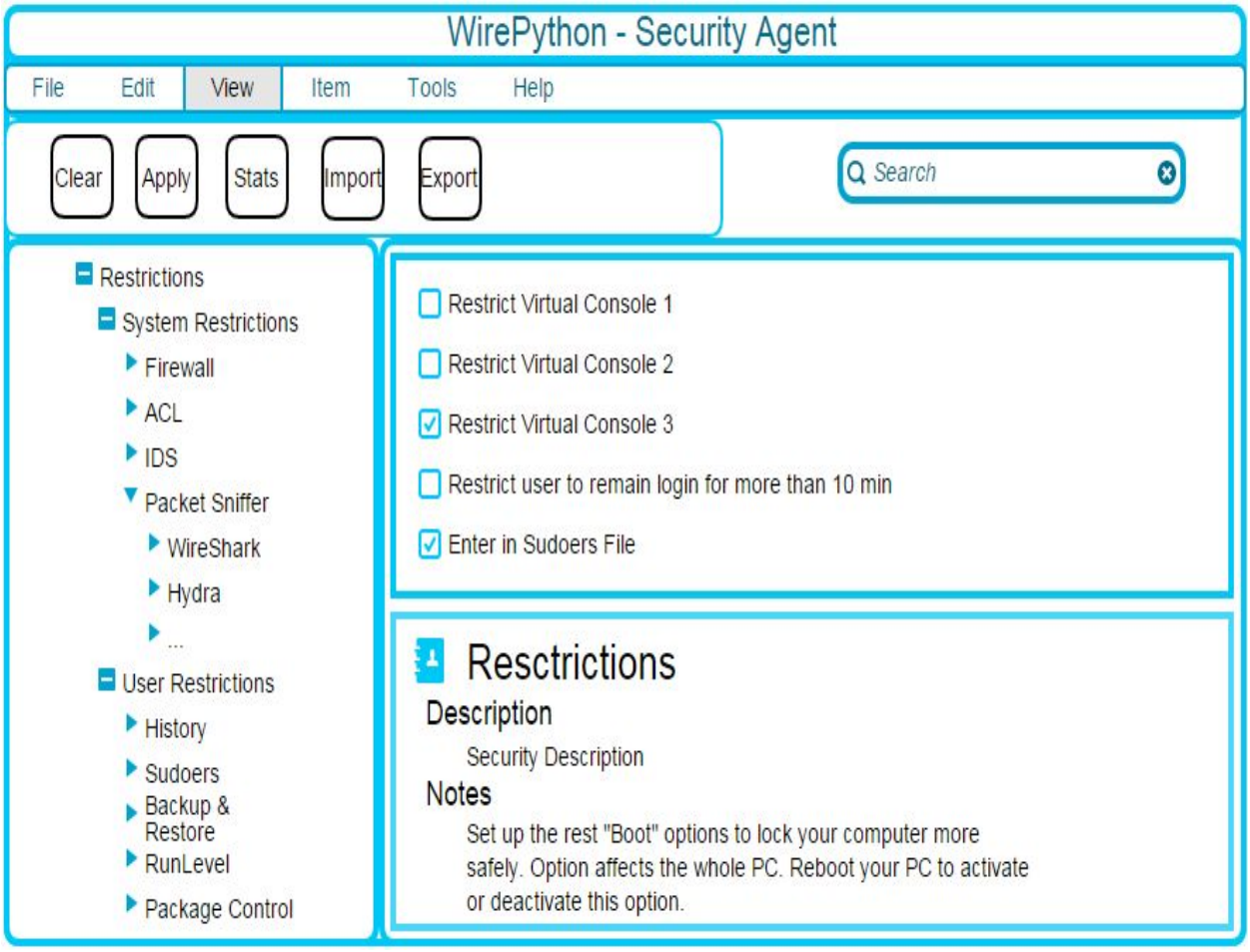
- IDS (Intrusion Detection System)

  This tool functionality help detect any type of attack and details about it like, how, when and where did the attack occurred.

- Packet Sniffer

  A tool like WireShark which can sniff and analyse packets and represent them to user, under the roof of "1st Security Agent"

EXPECTED LAYOUT:



Expected Layout for “GUI based Security Agent for Linux”

CHAPTER-7

REFERENCES & BIBLIOGRAPHY

- https://www.python.org/
- https://www.linux.com/
- http://www.linuxfoundation.org/
- https://wiki.python.org/moin/PyQt
- Wikipedia
    - https://en.wikipedia.org/wiki/Python_(programming_language)
    - https://en.wikipedia.org/wiki/PyQt
    - https://en.wikipedia.org/wiki/MySQL
    - https://en.wikipedia.org/wiki/SQL
    - https://en.wikipedia.org/wiki/Linux
- Books
    - Linux- The complete reference by Richard Peterson
    - O'Reilly "Programming Python"- by Mark Lutz